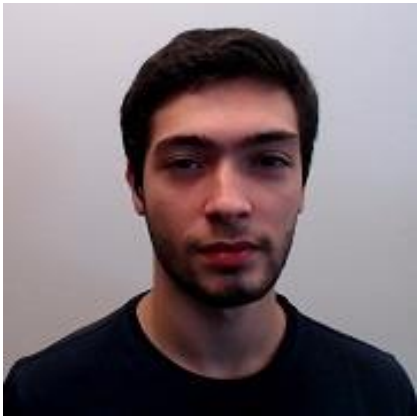
Universidade do Minho

[20-21]

Sistemas Operativos

MIEI

Grupo 19



Diogo Miguel da Silva Araújo,
A93313

Índice

Organização do projeto

O projeto realizado por este grupo alicerça-se, além dos ficheiros destinados ao cliente e ao servidor, "*aurras*" e "*aurrasd*", respetivamente, em outros 3, cada um destinado a um propósito específico.

Ora, começando pela base de todos, temos o "*ReadFilters*", onde estão definidas estruturas capazes de armazenar informação sobre os filtros necessários à realização do programa objetivo, assim como métodos para as manipular. Por sua vez, o ficheiro "*InputHandler*" foca-se na validação e filtragem dos comandos enviados pelo cliente e, por fim, no "*Communication*" encontram-se maneiras de, além de comunicar nas duas vias, isto é, Cliente-Servidor e Servidor-Cliente, também é onde se faz uma nova filtragem dos pedidos do utilizador, assegurando os resultados pretendidos para cada.

ReadFilters

Neste módulo estão presentes funcionalidades simples de "*get's*" e "*set's*" aplicados às estruturas criadas para assegurar a permanência da informação de cada filtro no sistema.
A estrutura final utilizada para esse efeito é um "*filters_folder*", onde ficam armazenados, num *array*, todos os filtros encontrados e carregados pelo sistema.

```c
// /----------------------------------//
typedef struct filter{                //
                                      //
    char* identifier;                 //
    char* executable;                 //
    unsigned int max_per;             //
    unsigned int in_use;              //
} filter;                             //
// /----------------------------------//
```

Estrutura de 1 filtro

```c
// /----------------------------------//
typedef struct filters_folder{        //
                                      //
    FILTER* filters;                  //
    unsigned int number_of_filters;   //
} filters_folder;                     //
// /----------------------------------//
```

Estrutura com todos os filtros

Quanto à leitura e respetivo parse do ficheiro *config*, que contém a informação sobre cada filtro, foi construído um processo de leitura por blocos de informação, onde se verifica um cuidado acrescido pelo bom formato deste. Por outras palavras, ao ler blocos de um tamanho fixo é possível, e até provável, que se retenha informação "cortada" a meio. Para tal, faz-se uso da função "*lseek*" para assegurar um bom formato do final do bloco (assegura que a última linha do bloco tem um formato legível).

Communication

Vias de comunicação simples

No ficheiro intitulado de "*Communication*" estão presentes, de imediato, duas funções que se viram essenciais para a realização do projeto. Uma, *read_from_server()*, assegura ao cliente um momento de espera por alguma resposta por parte do servidor com quem comunica, através de um *named pipe*. Este é um processo simples onde, abrindo o terminal de comunicação, espera pela informação que necessita e volta a encerrar a mesma.

Em paralelo com esta funcionalidade, é também utilizada uma função capaz de automatizar o processo de escrita de uma mensagem para o cliente, por parte do servidor, utilizando a mesma via, apenas num sentido contrário.

```
// /-----------------------------------------------------------------------------------------------------------------
int write_to_cliente(char* something){


    int server_fd = open(Server_Client, O_WRONLY);
    simple_error_handler(server_fd, "\n#> [server_error]: Nao foi possivel criar ligacao com cliente\n");
    if(server_fd < 0) return -1;

    int write_status = write(server_fd, something, strlen(something));
    write(server_fd, "\n", 1);

    close(server_fd);

    simple_error_handler(write_status, "\n#> [server_error]: Nao foi possivel escrever para o cliente\n");
    if(write_status < 0) return -1;

    return 0;
}
// /-----------------------------------------------------------------------------------------------------------------
```

write_to_cliente, função responsável pelo envio de mensagens ao cliente

Tratamento do input do cliente

O outro propósito deste módulo é assegurar ao cliente a informação que este necessita, consoante os seus pedidos. Para isso, aos pedidos de status do servidor e filtros disponíveis no sistema, este consegue organizá-los de maneira prática, consoante o seu estado interno, e, através das vias de comunicação explícitas no ponto anterior, procede ao seu envio. Deste modo, com um *write_to_cliente*, por parte do servidor, e um *read_from_server*, por parte do cliente, é extremamente simples a partilha de informação.

```
if(!strcmp(argv[1], "status") || !strcmp(argv[1], "filters")){

    read_from_server();

    return 0;
}
```

Procedimento, por parte do cliente, aquando de um pedido "status" / "filters"

Por fim, ainda focando no tratamento dos pedidos do cliente, é importante referir que também é realizada uma filtragem dos filtros pedidos aquando de um "transform". Com isto, entenda-se uma verificação de todos os filtros que o utilizador requiriu, ou seja, o pedido só se realiza se todos os filtros existirem nos dados do servidor.

Aurras (cliente)

O espaço reservado à execução do modelo do cliente é bastante compacto e simples.

Como suporte para o seu funcionamento, é criado um terminal de comunicação via *named pipes,* com o único propósito de escrita, visto que é quem faz os pedidos. Para tal, parte-se do princípio de que o servidor já se encontra no seu estado "online", caso contrário, interrompe de imediato o cliente, avisando-o do sucedido.

```
signal(SIGINT, ctrl_c_handler);

int communication_fd = open(Client_Server, O_WRONLY);

simple_error_handler(communication_fd, "\n#> [offline]: the server seems to be offline\n\n\n");
```

Abertura do terminal de comunicação com o servidor

De seguida, e após uma filtragem de comandos realizada pelo módulo "InputHandler" (apenas verifica o formato dos pedidos), é construída uma única linha de texto contendo os argumentos que o cliente utilizou para o pedido. Viu-se necessário realizar esta junção porque, por vezes, ao realizar uma escrita dos argumentos separada, era difícil captá-los no formato pretendido (por ex: a leitura do servidor captava, por vezes, argumentos na mesma linha, outras vezes, apenas um argumento por linha).

Por fim, e assumindo um pedido do tipo "*transform*", por parte do servidor, é preparado um espaço para o pedido do mesmo e a espera do seu resultado. Enquanto não for avisado pelo servidor de que o seu processo começou a ser processado, este fica em espera.

```
if(control == 1){

    printf("\n#> pending...\n");
    fflush(stdout);
    read_from_server();

}
```

O transform é identificado pelo cliente pelo resultado [1] proveniente da filtragem do input

Aurrasd (servidor)

Variáveis globais de auxílio :

FILTERS_FOLDER stored_filters

Escolheu-se guardar a estrutura com os filtros carregados pelo sistema como uma variável global pelo abrangimento que isso proporciona. Até boa parte da construção do projeto, não foi utilizada como tal, porém, aquando do tratamento das tarefas em execução, isto é, atualizar os filtros aquando da sua utilização e aquando do término do uso dos mesmos, viu-se necessária tal implementação. Deste modo, tornou-se mais simples tratar os mesmos, através dos sinais SIGCHLD que, doutra maneira, não conseguiria aceder à estrutura dos filtros.

pid_to_update[N_EXECS][N_FILTERS]

A única maneira encontrada para seguir os processos envoltos da execução do programa envolvia o armazenamento dos seus *process identifiers.* De outro modo, não seria possível rastrear os processos inferiores criados durante a execução do servidor, nem quando estes terminavam, algo que se tornava imprescindível para controlar e atualizar os filtros em uso.

614	**0**	**2**	**0**	**1**	**0**

De maneira a explicar o array criado, atente-se na tabela acima criada. No primeiro elemento, é encontrado o *PID* associado a um processo que foi criado para executar um transform. Assumindo que o servidor está a alocar 5 filtros distintos, encontra-se, à sua direita, 5 espaços para esse efeito. Ora, aquando da verificação da disponibilidade dos filtros, atualiza-se a entrada deste *array* com o nº de filtros de cada tipo que o pedido continha, pela ordem em que estão armazenados na estrutura global para o efeito, mencionada no ponto acima.

Exemplo: Para um qualquer processo com *PID* 614, o pedido transform continha 2 filtros “alto”, e 1 “eco”.

char** tasks

Como última variável global a ser mencionada, resta este *array* destinado a armazenar, para cada processo criado na tabela do ponto anterior, uma *string* com o input do utilizador para cada *PID*. A sua identificação fica simplificada, pois é utilizado o mesmo índice da tabela anterior para alocar o mesmo, facilitando a manipulação de ambas as variáveis em sincronia.

Sig_child_handler :

Um procedimento usual para esperar que um processo filho/inferior termine é utilizar a *call wait().* No presente projeto, onde se pretende obter concorrência na execução de processos, tal via-se impossível.

Para resolver o problema, viu-se necessário utilizar uma *flag* disponível para o efeito pretendido, "*WNOHANG*", para indicar ao processo pai que não deve esperar que um processo termine, para continuar a sua execução. Assim, em vez de esperar, apenas reporta o estado de qualquer processo filho que tenha sido interrompido ou terminado.

De seguida, o seu procedimento baseia-se nos seguintes passos:

- Quando um processo filho termina, guarda o seu *PID*;
- Procura por esse *PID* no *array* global destinado para esse efeito;
- Se o encontrar, atualiza os filtros em uso com o auxílio dos valores guardados para esse efeito;
- Atualiza também as tarefas em execução (isto é, elimina-as, pois terminaram);
- Elimina as entradas na tabela de *PIDS*, pois já não serão utilizadas;

Procedimento de execução :

Ponto 1

Após verificar a disponibilidade de todos os filtros requeridos, o servidor trata de organizar, num array de *strings*, os nomes dos executáveis associados a cada filtro. Tendo os executáveis, o nome do ficheiro de entrada e o de saída, avança para o **ponto 2**.

Ponto 2

Neste momento, já num processo filho do servidor, é organizada a aplicação de cada filtro em separado, por ordem, a cada ficheiro destino. Se o servidor estiver a aplicar o primeiro filtro, tem atenção de o aplicar ao ficheiro de entrada. Quando há mais do que um, faz questão de os aplicar sucessivamente ao ficheiro de saída, apenas.

```
while(counter < n_filters){                                                    //

    int confirm = -1;
    if(counter == 0) {

        confirm = execute_aurras(filters[counter], inputfile, outputfile, 0);  //
    }

    else{

        confirm = execute_aurras(filters[counter], outputfile, outputfile, 1);  //
    }

    if(confirm == 0) counter++;
        else{ printf("\n#> [Server_status] : Erro no aurras!\n"); fflush(stdout); return -1;}
}
```

Estando todos os preparamentos terminados, resta apenas aplicar os filtros. Recorrendo a um novo processo filho, faz-se da prioridade abrir os ficheiros de input e criar os de output.

Obtendo sucesso na abertura dos ficheiros, é necessário atribuí-los como input e output ao programa que executará sobre eles.
Para tal efeito, através da call dup2(), atribuiu-se o ficheiro de entrada ao STDIN e o de saída ao STDOUT, ou seja, o programa utilizará o ficheiro de input e o resultado sobre ele será escrita sobre o do output.

```
dup2(input_fd, 0);
close(input_fd);
dup2(output_fd, 1);
close(output_fd);}
```

Atribuição dos file descriptors dos ficheiros de entrada e saída aos de STDIN, STDOUT, respetivamente

Dado por terminada a execução do programa de aplicação de filtros, o ficheiro final encontra-se disponível e, com a ajuda do sinal *SIGCHLD* enviado pelo término do processo, voltamos ao segundo ponto deste módulo, onde é aplicado o tratamento aos dados internos pelo *sig_child_handler*.

Conclusão

Dado por terminado o projeto, é difícil não referir que foi preciso haver muita pesquisa e fazer uso do método tentativa erro, até acertar. Foi um projeto interessante, deu para aplicar todos os conhecimentos adquiridos nas aulas e para entender muito melhor todos os conceitos que se devem dominar aquando do estudo desta cadeira.
Assim sendo, resta afirmar que considero ter obtido um bom resultado na realização do projeto.